**Report Project 1**

Department of Computer Science and Engineering, American University in Cairo

CSCE330401-Digital Design II

Ahmad Shalaby: ID: 900162698

Youssef Al-Qaffash: ID: 900192780

**Description**

In this Project, We have developed a simple simulated annealing-based placer that minimizes the total wire length.

**Overview**

Simulated annealing (SA) is a probabilistic method for approaching a function's global optimum. In particular, it is a metaheuristic to approximate global optimization for an optimization problem in a big search space. Even though simulated annealing typically only yields an approximation of the global minimum, it may be sufficient for many real-world issues when applied to extremely challenging computational optimization situations where exact techniques fall short. We used this way to implement our Cell Placement Tool

**Implementation language**

This Simulated-Annealing Cell Placement Tool was developed using python

**Assumptions**

*• HPWL (half-perimeter of the smallest bounding box containing all pins for a net) is used to estimate the*

*wirelength of any net*

*• The core area is an 2D array of empty squares (sites)*

*• Each cell is a square and matches the site size.*

*• The site size is 1x1.*

*• No site is assigned more than one cell.*

*• The distance between two cells is measured from the center of one cell to the center of the other*.

**Description of Implementation**

Firstly, our program takes the input file as a netlist this file has the following format

*The first line contains 4 values:*

*• The number of cells to be placed.*

*• The number of connections between the cells.*

*• The number of rows (ny) upon which the circuit should be placed.*

*• The number of columns (nx) upon which the circuit should be placed. I*

*Each of the following lines represents a net and it contains the following:*

*• The number of components attached to the net*

*• The list of components attached to the net*

Secondly, that our program starts to parse the txt file to extract the needed information such as the number of rows and columns

Thirdly, we make a random placement for the cells and start with the annealing algorithm as following:

*T = Tinit // Very high temp*

*Moves = 10 x (Number of cells)*

*while(T > Tfinal)*

*Pick 2 random cells and swap them*

*calculate the change in HWBL (ΔL) due to the swap*

*if(ΔL < 0)*

*then accept*

*else*

*reject with probability (1 - e-ΔL/T)*

*otherwise accept*

*if (count == moves):*

*T = 0.95 \* T*

*count = 0*

At the end, we print the outputs to the user and handle the GUI

**The Optimization steps:**

To increase the efficiency of our program, we created a way to not loop on all the connections from the giving input when we call our function HPWL, which get the total wire length by the summation of all half-perimeter wire length of each connection, the current placement after swapping. To do that, we created a dictionary that stores each cell and its locations in the input file. In other words, each cell will have values of where they are in the connections to get only the HPWL of these certain connections and update the total wire length. Another dictionary also stores each connection and its HPWL to help us update the HPWL of a particular connection and get the total wire length from this updated value. This lead to a decrease from O(N) to O(c), where c is a constant.

**Please note that:**

We have implemented 2 bonus features: which is displaying the final placement graphically using a GUI library called Tkinter.

And also we make our tool capable of updating the places of the cells while executing

We have tested our project intensively using the provided netlists.

**Challenges and Overcoming**

We faced a huge problem when we test on many cells like “d3.txt” file because the number of moves will be assigned to huge value which will lead to a huge delay in decreasing the temperature. This delay happens because for instance when the number of cells is 200 , then number of moves will be = 213 x 10 = 2130, so every 2130 times we will decrease the temperature by rate of 0.9 and this will lead to a huge delay as the initial temperature is very huge according to the equation which is = 500 x initial cost, which is estimate in our testing to be 500 x 3428 = 1714000 , and this to read to the final temperature which is estimated to 0.000083. Therefore, to reach to this value by decreasing the temperature every 2130 move will take a very long time.

We solved this problem by decrease the number of moves and that’s done by making

moves = 10 x number of cells / 25 , and that led to make the program runs faster than before but after about 30 seconds of delay when it deals with a huge number of cells and that delay increases a bit when number of cells increases more.

**Known Bugs or Issues**

No known bugs or issues in our simulator if the instructions are followed correctly.

**User Guide:**

1. To run this Simulated-Annealing Cell Placement Tool, you will may use any python compile of your choice and you will need to import a library called Thinker for the GUI.
   1. We have provided a folder (Tests). This folder include some sample net lists that you can use for testing. You will copy what is in those txt files and put them into the netlist.txt.
2. Compile and run the program.
3. When the program starts, it will show you the first random placement and the final output on the terminal and it will display the final placement graphically.
4. Please note:

In the GUI the empty place is represented in green and the used place is represented in yellow area with the number of the cell

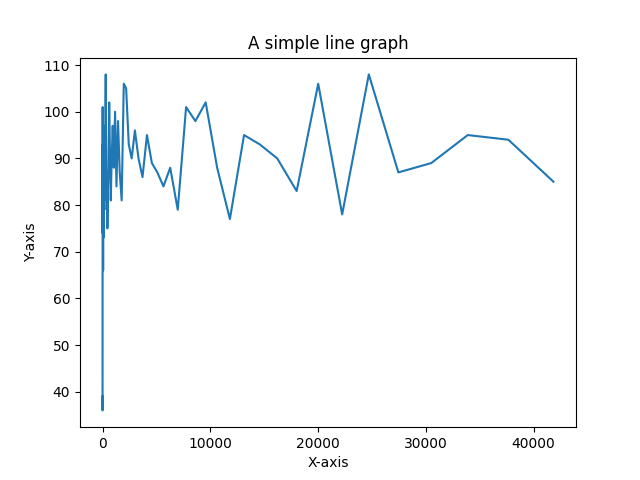
Graphs:

Temperature vs TWL

D0

Chart, line chart

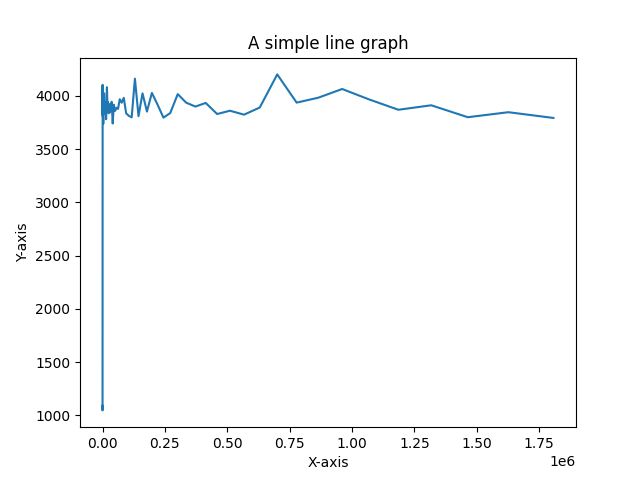
Description automatically generated



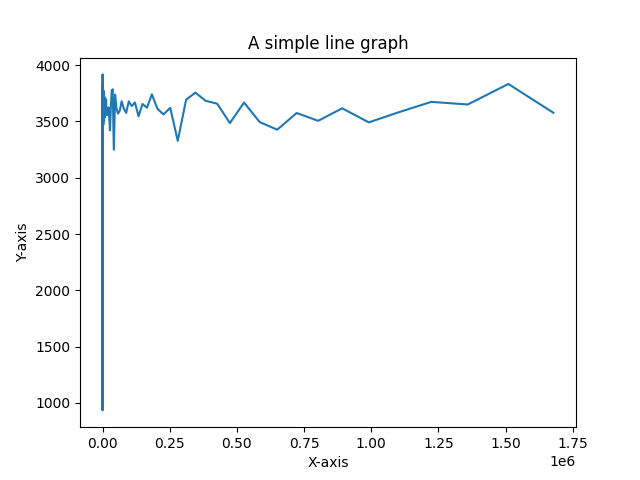
D0 D1

Chart, line chart

Description automatically generated

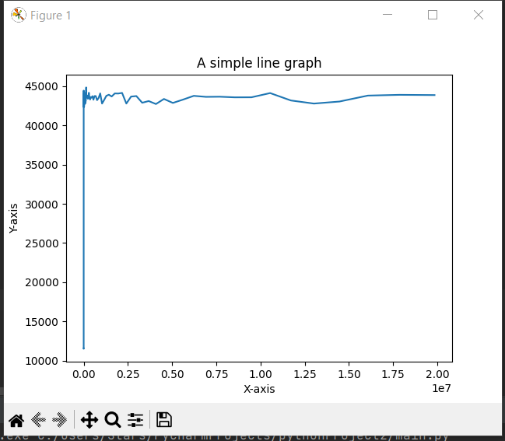


D2 D3

Chart

Description automatically generated

T1 T2



T3

Cooling Rate vs TWL

